From Sensors to Visualization Dashboards: Need for Language Composition

Sébastien Mosser, Ivan Logre, Nicolas Ferry, Philippe Collet

From Sensors to Visualization Dashboards: Need for Language Composition

Sébastien Mosser[1], Ivan Logre[1], Nicolas Ferry[2], and Philippe Collet[1]

[1] Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis
[2] SINTEF IKT, NSS Department, Oslo, Norway

Abstract. In the context of the Internet of Things, the SensApp platform is designed to collect data from sensors and support the building of associated monitoring dashboards. Bridging the gap between sensors and visualization involves up to eleven kind of models, from state machine modeling the behavior of a sensor to task diagrams modeling the actions of the end-user. This paper describes this case study, emphasizing the need for domain specific modeling language composition mechanisms to support the activity of modeling modern software-intensive systems.

1 Introduction: Modeling a Platform for the IoT

According to the *Internet of Things* (IoT) paradigm, *things* can exchange data with other *things*, creating a network of interconnected *things*. Things rely on *sensors* to collect data (*e.g.,* temperature sensors to regulate radiators in *smart* buildings, probes in smartphones pushing statistics to phone companies).

In another context, means of transport with accurate sensors (*e.g.,* speed, location) can log context information about a given travel. Data are then pushed to applications that produce innovative services on top of this information (*e.g.,* air quality monitoring). Data might also be stored for data-mining purpose or correlation between data collected *on the fly* and older data sets. For example, the bike depicted in Fig. 1a holds 12 sensors and 2 cameras[3]. The collected data are used to gather information about the city surroundings. Eventually, all these data need to be visualized. Fig. 1b represents a monitoring dashboard associated to the bike. It aggregates the video feed obtained from the front camera (top-right) with the bike position displayed on a map (bottom-right). It also displays the altitude, heading and speed of the bike, with both instant values (bottom-left) and graph representation of past data (top-left).

In this context, SINTEF and I3S have been developing the SensApp platform [1] since 2010. It exploits the cloud-computing paradigm to implement a scalable platform able to collect and store data, eventually consumed by third-party service to build monitoring dashboard. An overview of the platform is represented in Fig. 2, where arrows represents the dataflow associated to the collected data. Things send data to the SensApp platform, hosted in a cloud. Data are then stored in dedicated databases, and forwarded to specialized dashboards that subscribe to given sensor feeds. These applications consume the data, in real-time for monitoring purpose or asynchronously for analysis purpose.

[3] `http://www.youtube.com/watch?v=kia5Vkx59nY`

(a) Enhanced bike (b) Monitoring dashboard

Fig. 1: Bike equipped with sensors, and its monitoring interface

In this paper, we present the SensApp platform from a modeling point of view, based on our experience and using state of the art *Domain-Specific Modeling Languages* (DSMLs). This paper is mainly descriptive, reporting facts obtained from this real-life application used in several EU projects. Our objective is to sketch a complete case study for DSMLs composition that relies on a application used by several research teams and industrial partners. We first describe the different DSMLs that are well-suited to capture the different concerns involved in the SensApp architecture (Section 2). We then report on our first analysis on the kind of relationships identified between the used DSMLs (Section 3).

2 Proliferation of Modeling Languages

The definition of the complete SensApp platform led to tackling different challenges, at different levels of abstraction and in completely different domains. The systematic use of DSMLs for each concern of the SensApp platform involves the definition of models in eleven different languages. We briefly describe each concern and DSML, starting from the ones that are closer to *things* and moving up to models related to monitoring dashboards.

Graph models. Sensors and things are organized in networks to collect data. These networks are modeled as graphs, where each vertex is a sensor and edges represent communication channels between sensors. This is captured through a basic and *ad hoc* graph meta-model. Models are notably used to formally ensure properties on the sensor network (*e.g.,* assess gossip protocols used for data propagation).

Thing's behavior models. To implement the behavior of a given thing, we used a part of the ThingML [2] DSML. This formalism is a combination of architecture models, state machines and an imperative action language. These models are exploited to *(i)* generate code executed on microcontrollers and *(ii)* simulate the modeled behaviors for verification purpose.

Communication models. Sensors communicate with SensApp according to different protocols (*e.g.,* bluetooth, radio). An architecture model is needed to properly assemble the different gateways and generate the relevant glue

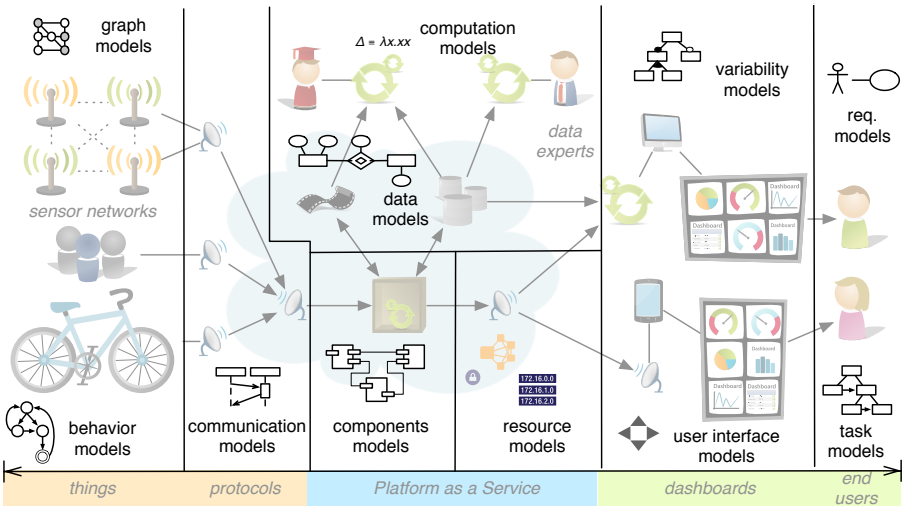

Fig. 2: Proliferation of Domain-specific Modeling Languages in SensApp

code on the intermediate layers. We used ThingML architecture description language to achieve this task.

Component models. Deployed SensApp instances must be frequently reconfigured to handle overload of incoming data. SensApp thus relies on a modular implementation and the Kevoree component middleware [3] is used to reify software component and assemble them through *models at runtime* techniques. Kevoree models are dedicated to clouds, and can be adapted at runtime, supporting the dynamic reconfiguration of a given assembly.

Data models. The modeling of data is essential for SensApp, as data are the core of the platform. Data and their encoding are modeled according to the SENML [4] formalism, and stored in databases relying on different paradigms (*e.g.,* document-oriented, journalized datasets, video blobs). Class diagrams are used to model SENML data types, and model mappings are used to assess the proper transformation between SENML and its storage-specific encoding (classical PIM $\rightarrow$ PSM transformation).

(Data) Computation models. Stored data are exploited by data miners to extract knowledge from the collected datasets. As sensors might produce tremendous amount of data, dedicated languages are used to express such data handling processes. The Pig DSL [5] supports the definition of these processes in a declarative way.

Resource models. In order to deploy and operate SensApp, we use the CloudML [6] environment. It provides a DSML along with a run-time environment that facilitates the specification of provisioning and deployment concerns of multi-cloud systems at design-time and their enactment at run-time.

Requirement models. The way data are used dramatically differs from one class of users to another. For example, a diabetic person needs to be alerted when her blood glucose concentration is too high, whereas a physician has

to monitor the evolution of such a concentration during a given period of time. UML Use Case diagrams are used to model these actors and their interactions with the monitoring interfaces.

Task models. For each use case, a detailed scenario is modeled, ordering the different tasks to be achieved for this use case. Task models are classically used by the HCI community, for example using the CTT formalism [7] to organize hierarchically and temporally the tasks to be done.

User Interface models. Widgets used to visualize data need to be organized in the final dashboard, according to two dimensions: *(i)* the spatial organization of widgets in a given frame, and *(ii)* the sequential organization of frames between each others. User interface models are designed for this purpose, according to the CAMELEON reference framework [8]. This framework does not provide any DSML reference implementation, so *ad hoc* languages are defined "according to" it.

Variability models. While building a monitoring dashboard, software engineers are in front of large collections of widgets that can be used for the implementation of a given task. Considering that variability models support the reification of this diversity, we used the Familiar DSML [9] to address this point. These models are exploited to check logical constraints, preventing an engineer to select, when configuring the dashboard, inappropriate widgets according to a given task.

3 Field Experience: Need for Language Composition

The previous section described each DSML used in the case study independently. Based on this case study, we identified two scenarios of composition between DSMLs. First, DSMLs need to be integrated one with each others, for example when one language is used to implement a model element of another one. Secondly, DSMLs must support co-evolution, typically when the two domains are really different. We illustrate these two classes of composition based on two scenarios identified in the SensApp example.

Integration: From Sensors to the Cloud. The communication models used to reify the way sensors communicate with the cloud platform are critical from a verification & validation point of view. These models are used by the sensor network architectural model (a graph), and also by the component model used to reify the cloud application. It is then critical that each tiers involved in the communication *shares* the same communication model. As a consequence, the architect of the sensor network must interact with concepts from the communication domain, shared with the cloud expert who distribute the different software components in a cloud architecture.

Co-evolution: Bridging the gap Between Visualizations and Sensors. Task models are designed by ergonomists, where data models are handled by database experts. These two domains are too far from each other to imagine the share of concepts. But the associated models are closely related: one cannot realize a given task if the expected data are not provided (*e.g.,* monitoring the diabete threshold is not possible without the availability of the

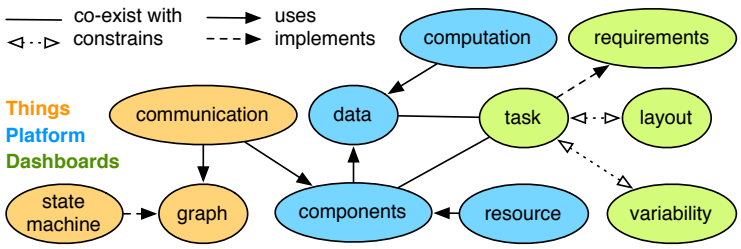

Fig. 3: Composition of Domain-specific Modeling Languages

blood sugar concentration level in the data model). Thus, a way to identify the relationships between heterogeneous model elements must be defined, as well as appropriate mechanisms to automatically compute the impact of the evolution of one model over another.

From these scenarios, we refine four kinds of relationship between the different languages. We represent in FIG. 3 the eleven DSMLs used in the SENSAPP case study and their relationships. We see the definition of these relationships as an interesting research line. As a first step, we illustrate each one on concrete examples in the following list.

- *co-exists with.* The component model used to reify the cloud application is loosely coupled with the task models defined for each user role. We expect these models to be coupled by a small and well-defined interfaces.
- *uses.* The component model uses the data model that reify the sensed data. In this case an evolution of the data model impacts the component one.
- *constrains.* The way end user tasks are organized in the task model will disallow several layout configurations that cannot implement it. As a consequence, the task model constrains the expressiveness of the layout ones. This interaction is bi-directional, as a given layout will also disallow several task organizations. In our case study we observe a bi-directional nature in each occurrence of this relationship.
- *implements.* A task diagram actually implements a given use case. Several alternative task models can be used to implement the same use case.

4 Summary

In this paper, we described the SENSAPP platform, dedicated to collect data from the IoT. It includes eleven DSMLs used to model each part of the system. Based on this case study, we described the need for composition of these languages at the modeling level, to support both integration and co-evolution of the designed models. Actually, the definition of all these models was cumbersome, time-consuming and trigger challenging maintenance issues. Thus, without any composition support provided at the DSML level, the approach advocating the systematic use of domain-specific artefact did not fit such a large case study. However, we do believe that each domain of SENSAPP needed to be modeled, with respect to the four intentions of the modeling activity: *(i)* to abstract, *(ii)*

to reason about, *(ii)* to document and finally *(iv)* to transform [10]. This case study emphasize the need for DSMLs composition mechanisms. Ongoing work consists in building the complete SensApp platform, while carefully observing and listing the compositions that are implemented between *ad hoc* and reused DSMLs. We thus plan to expose SensApp as a precisely defined case study for DSML composition issues. We will then focus on the relationships between specific models such as the interactions between the data, task and layout models in order to design adapted visualizations of sensor data from user task.

Acknowledgments. This work is partially supported by the IDOL project (PHC Aurora #28864TK) and the STM3 (Solution for the Treatment and Monitoring in Mobile Medicine, `http://www.pole-scs.org/projets?letter=S&page=5`) project. Authors want to thank Antoine Pultier for his work on the visualization platform, Brice Morin and Anne-Marie Déry for their feedback.

References

1. Mosser, S., Fleurey, F., Morin, B., Chauvel, F., Solberg, A., Goutier, I.: SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services. In: Management of Resources and Services in Cloud and Sky Computing (MICAS), Timisoara, IEEE (September 2012)
2. Fleurey, F., Morin, B., Solberg, A., Barais, O.: MDE to Manage Communications with and between Resource-Constrained Systems. In: MoDELS'11: 14th International Conference on Model Driven Engineering Languages and Systems, Wellington, New Zealand, October 16-21, 2011. Proceedings
3. Fouquet, F., Daubert, E., Plouzeau, N., Barais, O., Bourcier, J., Jézéquel, J.M.: Dissemination of Reconfiguration Policies on Mesh Networks. In Göschka, K.M., Haridi, S., eds.: DAIS. Volume 7272 of LNCS., Springer (2012) 16–30
4. Jennings, C., Arkko, J., Shelby, Z.: Media Types for Sensor Markup Language (SENML) (2012)
5. Gates, A., Natkovich, O., Chopra, S., Kamath, P., Narayanam, S., Olston, C., Reed, B., Srinivasan, S., Srivastava, U.: Building a highlevel dataflow system on top of mapreduce: The pig experience. PVLDB **2**(2) (2009) 1414–1425
6. Ferry, N., Rossini, A., Chauvel, F., Morin, B., Solberg, A.: Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In: CLOUD 2013: IEEE 6th International Conference on Cloud Computing, IEEE Computer Society (2013) 887–894
7. Paternò, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In Howard, S., Hammond, J., Lindgaard, G., eds.: INTERACT. Volume 96 of IFIP., Chapman & Hall (1997) 362–369
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers **15**(3) (2003) 289–308
9. Acher, M., Collet, P., Lahire, P., France, R.B.: Familiar: A domain-specific language for large scale management of feature models. Sci. Comput. Program. **78**(6) (2013) 657–681
10. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). Addison-Wesley Professional (2005)